

Timothée Gauthier  
Thomas Huot-Marchand

# Utilisation d'une Wiimote pour contrôler une brique NXT à travers internet

Tuteur : Mr Millet  
Année 2009/2010

## **Présentation**

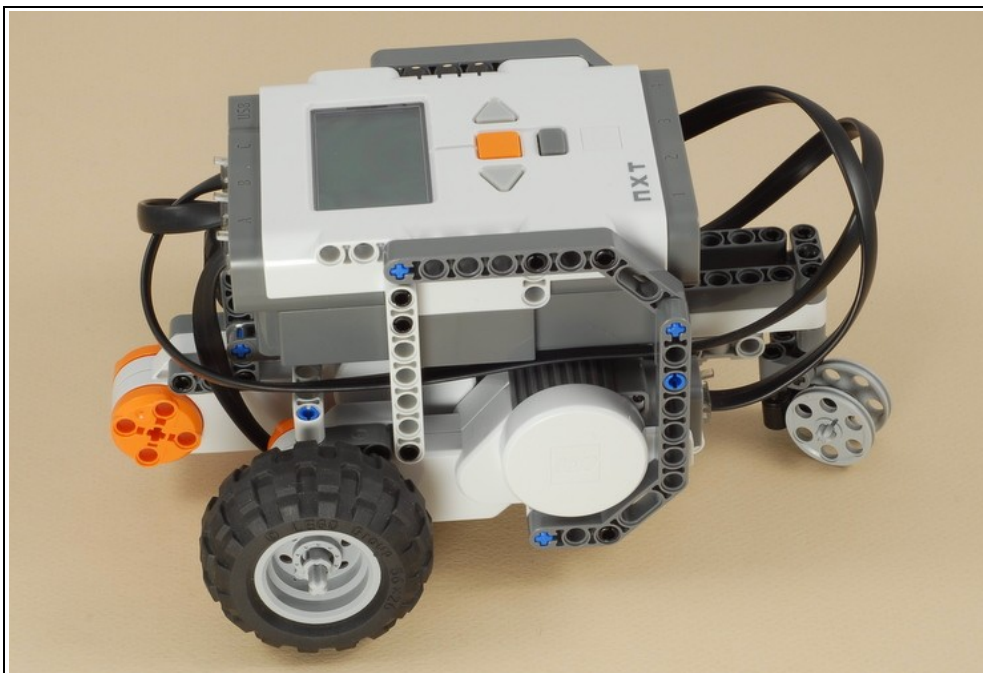
Notre projet consiste à développer une application afin de commander un robot fabriqué en brique Lego Nxt en passant par internet avec une Wiimote. Le langage de programmation utilisé est le JAVA.

## Table des matières

I . Pré-requis.....	4
II . Démarrage rapide.....	5
III . Choix du langage.....	5
IV . Détail du programme :.....	6
1 . Librairies utilisées dans le programme: .....	6
2 . Description des différentes classes.....	7
a) Start.class.....	7
b) Choix.class.....	7
c) Serveur_Brick.class.....	8
d) Fenetre.class.....	9
e) Ir.class.....	10
f) Wiimote01.class.....	10
Connexion.....	10
Réception et interprétation de l'Infrarouge.....	10
Réception et interprétation des accéléromètres.....	11
interprétation des boutons.....	11
Limitation du nombre de commandes.....	12
g) Client.class.....	12
h) Constantes.class.....	12
i) Brick.nxj.....	12
V . Description des commandes de la Wiimote.....	13
VI . Le Streaming .....	14
1 . Configuration pour la diffusion:.....	14
2 . Configuration pour la réception:.....	14
VII . Améliorations possibles:.....	14
VIII . Truc et Astuces.....	15
IX . Annexes.....	16
1 . Start.java.....	17
2 . Choix.java.....	18
3 . Serveur_Brick.java.....	20
4 . Constantes.java.....	22
5 . Client.java.....	23
6 . Fenetre.java.....	24
7 . Ir.java.....	27
8 . Wiimote01.java.....	29
9 . Brick.java.....	36

## I. Pré-requis

- Un robot en brique NXT dont le modèle de construction est disponible à cette adresse:  
[http://nxtprograms.com/3-motor\\_chassis/index.html](http://nxtprograms.com/3-motor_chassis/index.html)
- Le firmware LEJOS beta 0.8.5 remplaçant le firmware original de LEGO
- Une webcam relié à l'ordinateur coté brique NXT
- Deux PCs avec le Bluetooth et connexion réseaux
- Une Wiimote
- Java JRE, VLC, Bluez (linux), WIDCOMM ou BLUESOLEIL (Windows)



*Illustration 1: Modèle de la brique NXT du projet*

## II . Démarrage rapide

- 1) Allumer la brique et lancer le programme brick.nxj
- 2) Lancer Wiimote\_internet\_NXT.jar sur le pc côté brique et choisir « côté brique »
- 3) Lancer VLC sur les deux ordinateurs (voir partie streaming pour la diffusion)
- 4) Lancer Wiimote\_internet\_NXT.jar sur le pc côté Wiimote et choisir « côté Wiimote », attention à bien choisir le même port sur les 2 PCs et à bien inscrire l'adresse ip du serveur.
- 5) Connecter la Wiimote en appuyant sur 1 et 2.

## III . Choix du langage

L'intégralité du programme permettant de faire communiquer les deux appareils a été écrit en JAVA pour plusieurs raisons :

- Une seule et même archive fonctionnant sur les OS les plus répandus
- Il permet de créer facilement des programmes client-serveur
- La librairie Bluecove(\*) permet de faire fonctionner le bluetooth de façon fiable
- WiiRemoteJ.jar est une excellente librairie pour communiquer avec la wiimote

(\*) Pour que la librairie Bluecove puisse fonctionner les ordinateurs ont besoin des programmes suivant :

- Linux : Bluez (disponible dans les dépôts)
- Windows: Bluesoleil (déconseillé) ou Widcomm

## IV .     **Détail du programme :**

Le programme est composé de 8 classes:

- Start.class : permet de lancer le programme
- Constantes.class : donne les codes pour faire bouger la brique
- Choix.class: permet de spécifier les ports d'écoutes et de choisir si l'on veut lancer le programme côté Wiimote ou côté brique
- Serveur\_Brick.class : gère le flux de données entre la brique et le pc
- Fenetre.class : affiche les informations de communication entre le pc et la Wiimote
- Client.class : envoie les données sur internet, initialise la connexion et réserve les ports.
- Ir.class: permet de visualiser l'emplacement du point infrarouge.
- Wiimote01.class : reçoit et interprète les données de la Wiimote, permet aussi d'envoyer certains ordres (vibrations, LED, son)

### **1 .   *Librairies utilisées dans le programme:***

com.intel.bluetooth.BlueCoveConfigProperties;

permet d'initialiser le bluetooth et d'utiliser bluecove.jar qui dialogue avec la pile bluetooth déjà installée.

java.net.\*; permet de gérer les accès réseaux.

wiiremotej.\*; permet de dialoguer avec la Wiimote

wiiremotej.event.\* ; déclenche les événements de la Wiimote.

javax.swing.\*; permet d'afficher des interfaces graphiques.

java.awt.\*; utilisée pour dessiner le Jpanel de l'infrarouge.

java.io.\*; permet de gérer les entrées-sorties (accès fichiers, gestion de répertoires,...)

lejos.pc.comm.\*; pour communiquer par bluetooth entre la brique et le pc

java.util.date\*; permet de récupérer le temps (en ms) pour ne pas exécuter deux commandes en même temps.

## 2 . Description des différentes classes

### a) Start.class

C'est la class de démarrage, elle permet, une fois le choix effectué d'initialiser la classe Client Fenetre et Wiimote01 ou seulement Serveur\_Brick.

### b) Choix.class

Contient une interface graphique permettant de sélectionner le rôle du pc: s'il sera le pc connecté à la brique ou bien celui connecté à la Wiimote.

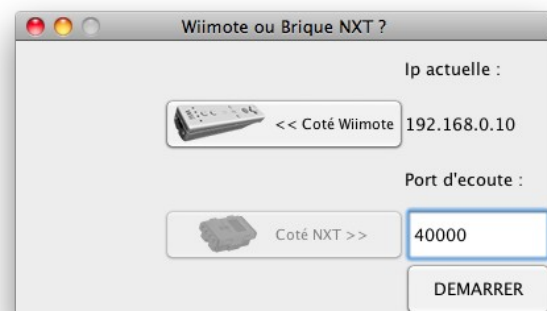


*Illustration 2: Fenêtre principale obtenue au lancement du programme, il faut sélectionner la fonction de l'ordinateur*

Si l'ordinateur est connecté à la Wiimote, il faut régler le port et l'adresse Ip d'envoi au serveur.



*Illustration 3: côté Wiimote sélectionné*



*Illustration 4: côté Brique NXT sélectionné*

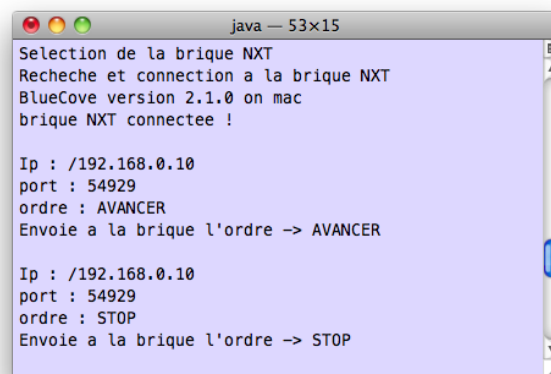
Si l'ordinateur est connecté du côté de la brique, il faut régler le port d'écoute.

L'adresse ip et le port doivent donc être identique côté brique et côté Wiimote.

Ensuite il faut cliquer sur démarrer, la visualisation du serveur se fait dans un terminal (il faut donc exécuter le programme comme cela : `java -jar Wiimote_internet_NXT.jar`)

Pour le programme qui gère la Wiimote, une fenêtre de contrôle s'ouvre.

### c) **Serveur\_Brick.class**



```
java — 53x15
Selection de la brique NXT
Recherche et connection a la brique NXT
BlueCove version 2.1.0 on mac
brique NXT connectee !

Ip : /192.168.0.10
port : 54929
ordre : AVANCER
Envoie a la brique l'ordre -> AVANCER

Ip : /192.168.0.10
port : 54929
ordre : STOP
Envoie a la brique l'ordre -> STOP
```

#### Description de la classe

Initialisation :

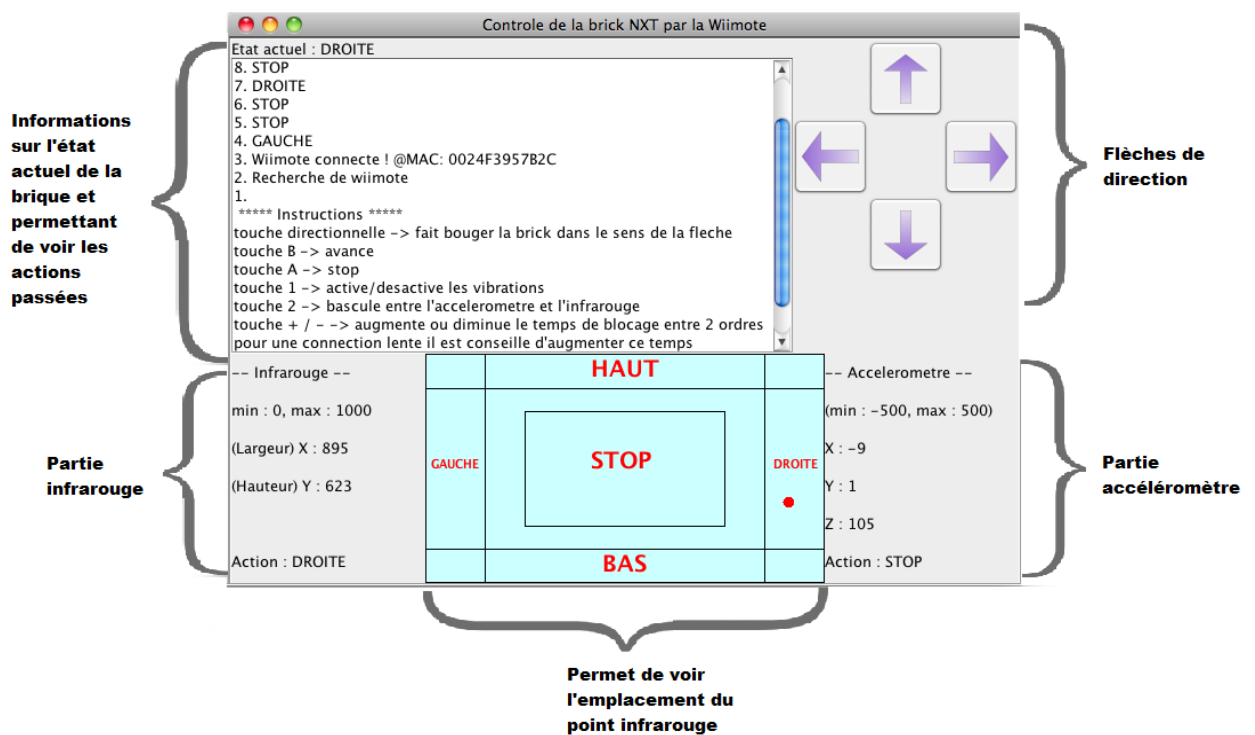
- du port d'écoute
- rechercher et se connecter à la brique
- les flux bluetooth entrants et sortants

Fonctionnement :

Le thread attend l'arrivée d'une trame, si c'est un ordre compréhensible, il l'envoie en bluetooth sinon il envoie la commande « Stop ».

la méthode « conversion » permet de traduire les ordres chiffrés en mots clairs.



d) **Fenetre.class**

*Illustration 5: Fenêtre obtenue lorsque l'ordinateur est connecté à la Wiimote*

## Description de la classe

Cette classe permet d'avoir une vue des données reçues par la wiimote ainsi que les actions envoyés sur internet.

Elle charge la classe « Ir » qui permet d'avoir un Jpanel montrant l'emplacement de la lumière infrarouge reçue par la Wiimote.

Fonctionnement :

Attend les actions de la wiimote ou les actions des boutons symbolisant des touches directionnelles.

Méthodes :

void set_l_info(String s)	Ajoute automatiquement le texte en paramètre dans le TextArea en incrémentant le repère et en déplaçant le curseur au début.
void set_l_acc_XYZ(int x, int y, int z)	Actualise les informations des labels de la partie « accéléromètre »
void set_l_acc_action(String s)	Actualise les « actions » déclenché par l'accéléromètre
void set_l_ir_xy(int x, int y)	Actualise les informations X et Y des labels de la partie «infrarouge»
void set_l_ir_action(String s)	Actualise les « actions » déclenché par la position de l'infrarouge.
void affichePointIR(double x, double y)	Envoie les nouvelles coordonnées du point infrarouges puis redessine le JPanel
mousePressed (MouseEvent e)	Déclenche les actions relatives aux flèches
mouseReleased (MouseEvent e)	Envoie l'ordre « STOP » à la brique

**e) Ir.class**

JPanel affichant l'emplacement de la source infrarouge par rapport à la wiimote. S'adapte en fonction de la taille de la fenêtre.

Les cadres représentant les déclenchements des actions sont dessinés d'après les valeurs utilisés dans la classe Wiimote01.

**f) Wiimote01.class**

Classe principale permettant la connexion à la wiimote, la récupération des événements (appui sur les boutons), la position des accéléromètres et la positions des sources infrarouges.

Fonctionnement de la classe :

**Connexion**

connecter\_wiimote() :

Affiche les commandes de la wiimote

recherche d'une wiimote

si aucune wiimote n'est trouvé, on recommence la recherche

si connectée, elle vibre et on initialise les données qu'elle devra envoyer en bluetooth

wiimote.addWiiRemoteListener (new WiimoteListener ()) ; envoie les boutons pressés

wiimote.setIRSensorEnabled (true, WRIREvent.BASIC) ; active la réception infrarouge. BASIC signifie qu'elle envoie uniquement la position des sources. (il existe d'autres modes pour récupérer l'intensité de la luminosité).

wiimote.setAccelerometerEnabled (true) ; envoie l'état des accéléromètres.

**Réception et interprétation de l'Infrarouge**

void IRInputReceived(WRIREvent ire); c'est grâce à cette méthode qu'est traité la réception de la source infrarouge :

on ne teste que la source 0 (possibilité de gérer jusqu'à 4 sources)

La position est multiplié par 1000 pour être plus facilement exploitable. Donc les valeurs vont de 0 à 1000.

Pour la source donnant la largeur (irX) on fait 1000-positionSourceX pour inverser le sens de déplacement.

Ces deux valeurs sont envoyé à la fenêtre de contrôle qui met à jour les informations et redessine le JPanel

Pour déterminer la commande à envoyer (avancer reculer gauche droite ou stop) on analyse la position de la source :

Y compris entre 0 et 151 exécute « haut »

Y compris entre 850 et 1000 exécute « bas »

X compris entre 0 et 151 exécute « Gauche »

X compris entre 850 et 1000 exécute « Droite »

Si X ou Y est compris entre 250 et 750 exécute « STOP »

### ***Réception et interprétation des accéléromètres***

On récupère l'accélération en X, Y et Z que l'on convertit en un Integer allant de -100 à 100, placé dans la variable `etat[]`.

Seul les états de X et Y sont testé pour déclencher les mouvements car la brique évolue sur un plan, donc en 2D.

Y supérieur à 60 exécute « haut »

Y inférieur à -60 exécute « bas »

X supérieur à 70 exécute « Gauche »

X inférieur à -70 exécute « Droite »

Pour chaque commande, on met une variable sur "true" pour signaler qu'il est inutile de relancer cette commande à la prochaine détection (qui est très rapide) tant que l'on n'est pas passé par STOP

Si X ou Y est compris entre -40 et 40 exécute on « STOP » et on réinitialise les variables sur false.

Pour l'interprétation des mouvements par accéléromètre ou infrarouge, les valeurs ont été prise arbitrairement, vous pouvez régler la sensibilité en changeant ces valeurs

### ***interprétation des boutons***

Il retient deux état : lorsque le bouton est pressé et relâché.

Pour la croix directionnelle :

Pressé : exécute la commande en fonction de la flèche

Relâché : exécute la commande stop

Bouton A : bouton de secours : envoie la commande STOP et stoppe les vibrations de la wiimote sans tester si elles sont activées ou non.

Bouton 1 : bascule entre le mode accéléromètre ou infrarouge en modifiant la variable `mode`.

Bouton 2 : Active / désactive les vibrations en modifiant la variable « vibrations »

Bouton + et - : Permettent de définir le temps d'attente obligatoire entre 2 commandes. En effet, si on secoue la wiimote ou si plusieurs sources infrarouges sont visibles ou si plusieurs touches sur la croix directionnelle sont enfoncées, le nombre de commande généré peut être très grand dans un laps de temps très court. Et si les trames UDP peuvent suivre selon le débit, le bluetooth du côté de la brique NXT ne sera pas assez rapide.

Le temps entre deux commandes est défini en milliseconde par la variable `temps_blocage`.

**Limitation du nombre de commandes**

Avant chaque exécution de commande (haut(), bas(), gauche(), droite()) on fait appel à la méthode is\_ok(). Cette méthode compare le temps écoulé entre 2 commandes (autre que STOP) et renvoie « true » si le temps est supérieur à temps\_blocage, « false » sinon.

En plus de la méthode is\_ok() il y a la méthode « vibre() » qui teste si les vibrations sont activées et réalise la même chose que is\_ok() mais avec les vibrations.

Cette fonction n'est pas bloquante : 2 commandes peuvent être envoyées rapidement et validées par is\_ok(), mais il est possible que la wiimote ne vibre pas à chaque commande.

Cette fonction a été rajoutée car de trop nombreuses vibrations ralentissaient fortement le programme.

disconnected() est déclenché lorsque la wiimote s'éteint (volontairement, ou non). Elle permet de relancer la méthode connecter\_wiimote() donc de connecter une nouvelle wiimote.

**g) Client.class**

Initialise et envoie les ordres dans des trames UDP en fonction du port et de l'adresse ip enregistré dans la première fenêtre du programme.

**h) Constantes.class**

Interface permettant de simplifier l'envoi des codes des commandes et le choix du mode entre accéléromètre ou infrarouge.

**i) Brick.nxj**

C'est le programme présent dans la brique NXT.

Fonctionnement :

Initialise sa connexion bluetooth et attend indéfiniment qu'un périphérique s'y connecte. Une fois connecté il entre dans une boucle testant « p ». p est le message reçu par bluetooth. Si p n'est pas un integer ou s'il y a un problème avec la connexion, p vaut « DECO » la brique réinitialise sa pile Bluetooth et recommence le programme depuis le début. Si p est un integer et p connu, les commandes sont traduites en rotations de moteurs.

Il implémente le même fichier « Constantes » afin de traduire facilement les ordres.

## V . Description des commandes de la Wiimote



Note: Plus le délai entre 2 informations transmises est petit, plus la transmission par bluetooth est difficile car lente.

Au maximum, le temps de latence entre 2 trames est de 1seconde et au minimum, de 100ms.

## VI . Le Streaming

Pour pouvoir voir les déplacements du robot à travers internet, il faut qu'une caméra soit fixée dessus, renvoie des images et que le pc côté Wiimote reçoive ces images. Pour la diffusion et la réception, nous utilisons vlc média player.

### 1 . Configuration pour la diffusion:

Média => Diffusion

Sélectionner l'onglet " périphérique de capture "

Mode de capture: DirectShow

Nom du périphérique vidéo: sélectionner la webcam branchée sur le robot

Cliquer sur Diffuser, puis suivant

Dans destination choisir HTTP puis ajouter

Entrer l'adresse IP du serveur et le port 80.

Dans option de transcodage cocher " activer le transcodage " et choisir le profil " video-DIV3 + MP3 (ASF)

Cliquer sur suivant et cocher " garder le flux de sortie actif ".

Puis diffuser

### 2 . Configuration pour la réception:

Média => Ouvrir un flux réseau...

sélectionner protocole HTTP

Dans le champ adresse entrer l'adresse IP du serveur et le port utilisé, en HTTP c'est le 80. Par exemple 127.0.0.1:80

## VII . Améliorations possibles:

Le programme peut être amélioré de différentes façons :

- Trouver un codec plus approprié pour le streaming
- Feedback de la brique
- Visualisation du parcours de la brique
- Utiliser le haut-parleur de la Wiimote
- Inclure la vidéo dans le programme Java
- Utiliser les Leds
- Mettre un pivot sur la webcam
- Webcam sans fil
- Sécuriser la connexion, et la passer en TCP

## VIII . Truc et Astuces

Compilation :

Pour le lancer en ligne de commande :

```
java -classpath ../lib/* Start
```

Bluecove ne tourne pas sur certains systèmes 64 bit, il est alors utile de rajouter -d32

```
java -d32 -classpath ../lib/* Start
```

Pour lancer Wiimote\_internet\_NXT.jar :

```
java -jar [-d32] Wiimote_internet_NXT.jar
```

N'oubliez pas de mettre le dossier « lib » dans le même répertoire que le jar

Pour compiler en ligne de commande le programme principal :

```
javac -classpath ../lib/* ./Start.java
```

Afin de réaliser la compilation et le lancement facilement il est conseillé de créer ce fichier bash et de l'appeler avec le nom de votre classe qui comporte le Main :

```
rm *.class
javac -cp ../lib/* $1.java && echo && echo lancement de $1 && java -cp ../lib/* $1
```

Pour créer le jar en ligne de commande :

```
jar cmvf manifest.txt Wiimote_internet_NXT.jar images/* *.class
```

Le fichier manifest.txt doit comporter ceci :

```
Main-Class: Start

Class-Path: lib/bluecove-2.1.0.jar lib/bluecove-gpl-2.1.0.jar lib/classes.jar lib/jtools.jar
lib/pccomm.jar lib/pctools.jar lib/WiiRemoteJ.jar
```

Si la Wiimote semble connecté en bluetooth mais pas au programme, allez dans les paramètres bluetooth de votre ordinateur et faite cliquer sur déconnecter Nintendo RVL-CNT. Appuyez de nouveau sur 1 + 2 sans relancer le programme.

La touche A de la wiimote permet d'envoyer la trame « stop » à la brique et de stopper les vibrations de la wiimote toute que toute.

Pour compiler pour la Brique NXT (Lejos doit être installé et configuré):

```
nxjc Brick.java
nxjlink -v Brick -o Brick.nxj
```

Pour l'envoyer à la brique nxt (bluetooth ou usb):

```
nxjupload Brick.nxj
```

La brique NXT doit avoir été synchronisée avec l'ordinateur avant le lancement du programme.

## IX .     **Annexes**

Liste des fichiers composant le programme principale.

Ces librairies doivent être présentent dans le dossier lib :

WiiRemoteJ.jar     classes.jar     pctools.jar     bluecove-2.1.0.jar     jtools.jar  
bluecove-gpl-2.1.0.jar     pccomm.jar

Vous les trouverez en téléchargeant Lejos (<http://lejos.sourceforge.net/>),

bluecove (<http://bluecove.org/>)

et la librairie pour la wiimote : <http://www.world-of-cha0s.hostrocket.com/WiiRemoteJ/>

ordre des sources :

Start.java  
Choix.java  
Serveur\_Brick.java  
Client.java  
Constantes.java  
Fenetre.java  
Ir.java  
Wiimote01.java  
Brick.java



## 1. Start.java

```

/*****
 *
 *   Timothée Gauthier
 *   Thomas Huot-Marchand
 *
 *   Projet tuteuré
 *   Année 2009//2010
 *****/

import com.intel.bluetooth.BlueCoveConfigProperties;
import java.net.*;

public class Start implements Constantes{

    public static void main (String[] args){
        //initialisation du bluetooth
        System.setProperty (BlueCoveConfigProperties.PROPERTY_JSR_82_PSM_MINIMUM_OFF, "true");

        //recupere l'adresse ip, fonctionne pour windows et mac, 127.0.0.1 sous linux
        String ip_local = new String("");
        try {
            InetAddress Ip = InetAddress.getLocalHost();
            ip_local = Ip.getHostAddress();
        }
        catch (Exception e) {
            ip_local = "Probleme";
        }

        Choix c = new Choix ("Wiimote ou Brique NXT ?", ip_local);

        while(c.ouverte()){ //on attend que l'utilisateur fasse le choix. c.ouverte() <- fonction qui return true si fenetre ouverte, false
            quand fermée
                try {Thread.sleep(300);} catch (InterruptedException e) {e.printStackTrace();}
        }

        if (c.choix_programme() == WIIMOTE){
            try{
                Client cl = new Client(c.getPort(), c.getIp()); //on transmet le port et l'ip du serveur entré dans la fenetre "Choix"
                Fenetre f = new Fenetre ("Controle de la brique NXT par la Wiimote", cl); // cree fenetre avec titre, et ip/port
                new Wiimote01 (f,cl) ;//on lui transmet la classe fenetre (pour afficher les actions) et la classe "envoyer"
            }
            catch (Exception z){
                z.printStackTrace();
            }
        }
        else{
            System.out.println("Selection de la brique NXT");

            try{new Serveur_Brick(c.getPort());}
            catch (Exception z){
                z.printStackTrace();
            }
        }
    }
}

```

## 2. Choix.java

```

/*****
 *
 *   Timothée Gauthier
 *   Thomas Huot-Marchand
 *
 *   Projet tuteuré
 *   Année 2009//2010
 *****/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Choix extends JFrame implements Constantes, ActionListener{

    JPanel jp = new JPanel();
    JPanel jp_east = new JPanel();
    JPanel jp_west = new JPanel();
    JPanel jp_choix = new JPanel();

    JPanel jp_config_brick = new JPanel ();
    JPanel jp_config_wiimote = new JPanel ();

    JButton jb_wiimote = new JButton ("<< Cot\u00e9 Wiimote", new ImageIcon(getClass().getResource("/images/wiimote.png")));
    JButton jb_brick = new JButton ("Cot\u00e9 NXT >>", new ImageIcon(getClass().getResource("/images/brick.png")));

    JTextField txt_port_brick = new JTextField ("40000");
    JTextField txt_port_wiimote = new JTextField ("40000");

    JTextField txt_ip_wiimote = new JTextField ("");

    JButton jb_start_brick = new JButton ("DEMARRER");
    JButton jb_start_wiimote = new JButton ("DEMARRER");

    String ip_local = new String("");
    String ip = new String("");
    int port = 40000;
    boolean f_ouverte = true; //pour passer à la suite
    int var_choix_programme ; //NXT ou WIIMOTE

    public Choix (String titre, String ip){

        super (titre);
        f_ouverte = true;
        ip_local = ip;
        txt_ip_wiimote.setText(ip_local);

        setDefaultCloseOperation (EXIT_ON_CLOSE);

        jp.setLayout (new BorderLayout());// GridLayout (1,3));
        this.add (jp);

        jp_choix.setLayout (new GridLayout (5,1) );
        jp_east.setLayout (new GridLayout (1,1) );
        jp_west.setLayout (new GridLayout (1,1) );

        jp.add(jp_choix, BorderLayout.CENTER);
        jp.add(jp_east, BorderLayout.EAST);
        jp.add(jp_west, BorderLayout.WEST);

        jb_brick.addActionListener (this);
        jb_wiimote.addActionListener (this);

        // choix principal wiimote ou brique
        jp_choix.add (new JLabel (""));
        jp_choix.add (jb_wiimote);
        jp_choix.add (new JLabel (""));
        jp_choix.add (jb_brick);
        jp_choix.add (new JLabel (""));

        jp_config_brick.setLayout (new GridLayout (5,1) );
        jp_east.add(jp_config_brick, BorderLayout.EAST);

        jp_config_brick.add (new JLabel ("Ip actuelle :"));
        jp_config_brick.add (new JLabel (ip_local));
        jp_config_brick.add (new JLabel ("Port d'ecoute :"));
        jp_config_brick.add (txt_port_brick);
        jp_config_brick.add (jb_start_brick);
        jp_config_brick.setVisible(false);

        jp_config_wiimote.setLayout (new GridLayout (5,1) );
        jp_west.add(jp_config_wiimote, BorderLayout.WEST);

        jp_config_wiimote.add (new JLabel ("Ip du serveur :"));
        jp_config_wiimote.add (txt_ip_wiimote);
        jp_config_wiimote.add (new JLabel ("Port du serveur :"));
        jp_config_wiimote.add (txt_port_wiimote);
    }
}

```

```

        jp_config_wiimote.add(jb_start_wiimote);

        jp_config_wiimote.setVisible(false);

        jb_start_brick.addActionListener(this);
        jb_start_wiimote.addActionListener(this);

        setResizable(false);
        pack();
        setLocationRelativeTo(null); //on centre
        setVisible(true);
    }

    public boolean ouverte(){
        //permet à la classe Start de savoir quand le choix est fait
        return f_ouverte;
    }
    public int choix_programme(){
        //permet à la classe Start de savoir quel choix est fait
        return var_choix_programme; //soit WIIMOTE soit BRICK
    }
    public int getPort(){
        return port; //pour initialiser la classe Client par la suite
    }
    public String getIp(){
        return ip; //pour initialiser la classe Client par la suite
    }

    public void actionPerformed(ActionEvent e){
        //TODO gérer les erreurs de port et ip.
        if (e.getSource() == jb_start_brick){
            dispose();
            port = Integer.valueOf(txt_port_brick.getText());
            var_choix_programme = BRICK;
            f_ouverte=false;
        }
        else if (e.getSource() == jb_start_wiimote){
            dispose();
            port = Integer.valueOf(txt_port_wiimote.getText());
            ip = txt_ip_wiimote.getText();
            var_choix_programme = WIIMOTE;
            f_ouverte=false;
        }
        else if (e.getSource() == jb_brick){
            jp_config_brick.setVisible(true);
            jp_config_wiimote.setVisible(false);
            jb_brick.setEnabled(false);
            jb_wiimote.setEnabled(true);
            pack();
        }
        else if (e.getSource() == jb_wiimote){
            jp_config_wiimote.setVisible(true);
            jp_config_brick.setVisible(false);

            jb_wiimote.setEnabled(false);
            jb_brick.setEnabled(true);
            pack();
        }
    }
}

```

### 3. Serveur\_Brick.java

```

/*****
 *
 *   Timothée Gauthier
 *   Thomas Huot-Marchand
 *
 *   Projet tuteuré
 *   Année 2009//2010
 *****/

import java.net.*;
import java.io.*;
import lejos.pc.comm.*;
import java.util.*;

public class Serveur_Brick implements Constantes{

    public Serveur_Brick(int port) throws IOException, SocketException{

        byte[] msg = new byte[2500];
        DatagramSocket s = new DatagramSocket (port);
        DatagramPacket p = new DatagramPacket (msg, msg.length); //l'objet ou le packet est stocké

        String mess_recu ;

        System.out.println("Recherche et connection a la brique NXT");

        NXTConnector conn = new NXTConnector();
        boolean connected = conn.connectTo("btspp://");

        if (!connected) {
            System.out.println("Impossible de se connecter de la brique NXT");
            System.exit(1);
        }

        System.out.println("brique NXT connectee !");
        DataOutputStream dos = conn.getDataOut(); //permet d'envoyer en bluetooth
        DataInputStream dis = conn.getDataIn(); //inutile ici, mais servirait pour un feedback venant de la brique

        int ordre = 0;

        for (;;){
            s.receive (p);

            mess_recu = new String(p.getData (), 0, p.getLength());

            //Si l'ordre reçu n'est pas un integer, on envoie "STOP" à la brique
            try{
                ordre = Integer.parseInt(mess_recu);
                mess_recu = conversion(ordre);
            }
            catch(Exception e){
                ordre = STOP;
                mess_recu = "Ordre reçu \" " + mess_recu + "\" inconnue !! "; //on affiche le paquet reçu du net si inconnu
            }

            System.out.println( "\nlp : " + p.getAddress() +
                                "\nport : " + p.getPort() +
                                "\nordre : " + mess_recu);
            p.setData(msg, 0 , msg.length);

            try {
                System.out.println("Envoie a la brique l'ordre -> " + conversion(ordre));
                dos.writeInt(ordre);
                dos.flush();
            } catch (IOException ioe) {
                System.out.println("IO Exception writing bytes:");
                System.out.println(ioe.getMessage());
                break; //si probleme avec le bluetooth, on quitte
            }
        }
    }

    private String conversion(int i){
        //convertit les ordre reçus d'integer en String, compréhensible pour les humains

        switch (i)
        {
            case UP:
                return UP_S;

            case DOWN:
                return DOWN_S;

            case LEFT:
                return LEFT_S;
        }
    }
}

```

```
    case RIGHT:
    return RIGHT_S;

    case STOP:
    return STOP_S;

    case DECO:
    return DECO_S;

    default:
    return "" + i; //si inconnu, on affiche l'ordre reçu.
  }
}
```

## 4. Constantes.java

```
/*
 *
 * Timothée Gauthier
 * Thomas Huot-Marchand
 *
 * Projet tuteuré
 * Année 2009//2010
 */
public interface Constantes {

    /* Listes des ordres//mouvements */

    final static int AVANCER = 10000;
    final static int UP = 10000;
    final static String UP_S = "AVANCER";

    final static int RECULER = 20000;
    final static int DOWN = 20000;
    final static String DOWN_S = "RECULER";

    final static int GAUCHE = 30000;
    final static int LEFT = 30000;
    final static String LEFT_S = "GAUCHE";

    final static int DROITE = 40000;
    final static int RIGHT = 40000;
    final static String RIGHT_S = "DROITE";

    final static int STOP = 50000;
    final static String STOP_S = "STOP";

    /* Fin de liste */

    final static int DECO = 60000;
    final static String DECO_S = "DECONNECTE";

    //pour la selection du programme à lancer
    final static int WIIMOTE = 10;
    final static int BRICK = 20;
    ////////////

    //utile pour l'affichage des valeurs de l'accéléromètre : etat[XX] etat[YY] etat[ZZ]
    final static int XX = 1;
    final static int YY = 2;
    final static int ZZ = 3;

    //choix du mode accelerometre ou infrarouge
    final static boolean ACCELEROMETRE = true;
    final static boolean IROUGE = false;
}
```

## 5. *Client.java*

```
/*
 *
 *   Timothée Gauthier
 *   Thomas Huot-Marchand
 *
 *   Projet tuteuré
 *   Année 2009//2010
 */

import java.net.*;
import java.io.*;

public class Client{

    String adresse;
    DatagramSocket soc;
    DatagramPacket pac ;

    public Client(int port, String ipp) throws IOException, SocketException, UnknownHostException{
        adresse = new String(ipp); //soit ip soit hostname
        soc = new DatagramSocket (); //selectionne un port libre sur la machine
        String temp_s = ""; //message à envoyer, ici l'argument
        byte[] temp_msg = temp_s.getBytes(); //conversion du message
        pac = new DatagramPacket (temp_msg, temp_s.length()); //crée le packet à envoyer
        pac.setPort(port);
        pac.setAddress(InetAddress.getByName (adresse));
    }

    public void envoyer(String s) {
        byte[] msg = s.getBytes(); //conversion du message
        pac.setData(msg);
        pac.setLength(s.length());

        try{
            soc.send (pac);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

    public void envoyer(int i) { //convertit l'integer en String pour l'envoi
        String s = new String (" " + i);
        envoyer(s);
    }
}
```

## 6. Fenetre.java

```

/*****
 *
 *   Timothée Gauthier
 *   Thomas Huot-Marchand
 *
 *   Projet tuteuré
 *   Année 2009//2010
 *****/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Fenetre extends JFrame implements Constantes, MouseListener{

    //(getClass().getResource("")) permet de retrouver les images dans l'archive .jar
    JButton jb_haut = new JButton (new ImageIcon(getClass().getResource("/images/img_haut.png")));
    JButton jb_bas = new JButton (new ImageIcon(getClass().getResource("/images/img_bas.png")));
    JButton jb_gauche = new JButton (new ImageIcon(getClass().getResource("/images/img_gauche.png")));
    JButton jb_droite = new JButton (new ImageIcon(getClass().getResource("/images/img_droite.png")));

    JLabel l_info = new JLabel ("Etat actuel : ");
    JTextArea text_info = new JTextArea (5,5);

    JLabel l_acc_X = new JLabel ("X : ");
    JLabel l_acc_Y = new JLabel ("Y : ");
    JLabel l_acc_Z = new JLabel ("Z : ");
    JLabel l_acc_action = new JLabel ("Action : ");

    JLabel l_ir_X = new JLabel ("(Largeur) X : ");
    JLabel l_ir_Y = new JLabel ("(Hauteur) Y : ");
    JLabel l_ir_action = new JLabel ("Action : ");

    Client envoi1;
    private int suivi_info = 0; //incrémentation du texte dans le textarea
    private Ir ir = new Ir(); //le JPanel qui affiche la position de l'infrarouge

    public Fenetre (String titre, Client client){

        super(titre);
        envoi1=client;

        JPanel jp = new JPanel();
        setDefaultCloseOperation (EXIT_ON_CLOSE);
        jp.setLayout (new BorderLayout ());
        add (new JScrollPane (jp));

        //panel d'info (un label plus le texte area)
        JPanel jp_infos = new JPanel ();
        jp_infos.setLayout (new BorderLayout ());
        jp.add (jp_infos, BorderLayout.CENTER);

        jp_infos.add (l_info, BorderLayout.NORTH);
        jp_infos.add (new JScrollPane(text_info), BorderLayout.CENTER); //texte scrollable
        text_info.setEditable(false);

        /*
        Panel des flèches de direction
        */
        JPanel jp_direction = new JPanel ();
        jp_direction.setLayout (new GridLayout (4, 3)); //4 pour équilibrer
        jp.add (jp_direction, BorderLayout.EAST);

        jp_direction.add (new JLabel (""));
        jp_direction.add (jb_haut);
        jp_direction.add (new JLabel (""));
        jp_direction.add (jb_gauche);
        jp_direction.add (new JLabel (""));
        jp_direction.add (jb_droite);
        jp_direction.add (new JLabel (""));
        jp_direction.add (jb_bas);
        //pour équilibrer
        jp_direction.add (new JLabel (""));
        jp_direction.add (new JLabel (""));
        jp_direction.add (new JLabel (""));
        jp_direction.add (new JLabel (""));

        jb_haut.addMouseListener(this);
        jb_bas.addMouseListener(this);
        jb_gauche.addMouseListener (this);
        jb_droite.addMouseListener (this);
    }

```



```

jb_haut.setForeground(Color.RED);
/*
Fin du panel de direction
*/

/*
Partie basse : infrarouge, accéléromètre
*/
//jp_bas -> infrarouge plus autres informations
JPanel jp_bas = new JPanel ();
jp_bas.setLayout (new BorderLayout()); //GridLayout(1,3));

//panel info_infrarouge X, Y
JPanel jp_ir_xy = new JPanel ();
jp_ir_xy.setLayout (new GridLayout (6,1));
jp_ir_xy.add(new JLabel("-- Infrarouge --"));
jp_ir_xy.add(new JLabel ("min : 0, max : 1000"));
jp_ir_xy.add(l_ir_X);
jp_ir_xy.add(l_ir_Y);
jp_ir_xy.add(new JLabel("  "));
jp_ir_xy.add(l_ir_action);
jp_ir_xy.setPreferredSize(new Dimension(170,200));
jp_bas.add (jp_ir_xy, BorderLayout.WEST);

jp_bas.add (ir, BorderLayout.CENTER); // le panel pour visualiser l'infrarouge

//panel info_accéléromètre X, Y, Z
JPanel jp_acc_xyz = new JPanel ();
jp_acc_xyz.setLayout (new GridLayout (6,1));
jp_acc_xyz.add(new JLabel("-- Accelerometre --"));
jp_acc_xyz.add(new JLabel("(min : -500, max : 500)"));
jp_acc_xyz.add(l_acc_X);
jp_acc_xyz.add(l_acc_Y);
jp_acc_xyz.add(l_acc_Z);
jp_acc_xyz.add(l_acc_action);
jp_acc_xyz.setPreferredSize(new Dimension(170,200));

jp_bas.add (jp_acc_xyz, BorderLayout.EAST);

ir.setPreferredSize(new Dimension(200,100));

ir.setBorder(BorderFactory.createLineBorder(Color.black));

jp.add(jp_bas, BorderLayout.SOUTH);

pack();
this.setSize(this.getWidth()+200,this.getHeight()+100); //évite les scrollbars en augmentant un peu la taille
setLocationRelativeTo(null); //on centre
setVisible(true);
}

public void set_l_info(String s){
/*
permet d'afficher et d'incrémenter le textearea_info
*/
suivi_info++;
l_info.setText("Etat actuel : " + s);
text_info.insert(suivi_info + ". " + s + "\n", 0);
text_info.setCaretPosition(0); //on place le curseur au début pour être toujours au dessus
}

/*
Affiche les information des accéléromètres ou de l'infrarouge
*/
public void set_l_acc_XYZ(int x, int y, int z){
l_acc_X.setText("X : " + x);
l_acc_Y.setText("Y : " + y);
l_acc_Z.setText("Z : " + z);
}
public void set_l_acc_action(String s){
l_acc_action.setText("Action : " + s);
}

public void set_l_ir_xy(int x, int y){
l_ir_X.setText("(Largeur) X : " + x);
l_ir_Y.setText("(Hauteur) Y : " + y);
}
public void set_l_ir_action(String s){
l_ir_action.setText("Action : " + s);
}

public void affichePointIR(double x, double y){ //met à jour le panel Infrarouge
ir.position(x,y);
ir.repaint();
}

public void mousePressed (MouseEvent e){
/*

```

```
on utilise mouse pressed et released pour contrôler plus facilement le start et stop
*/
if (e.getSource() == jb_haut){
    set_l_info(UP_S);
    envoi1.envoyer(UP);
}
else if (e.getSource() == jb_bas){
    set_l_info(DOWN_S);
    envoi1.envoyer(DOWN);
}
else if (e.getSource() == jb_gauche){
    set_l_info(LEFT_S);
    envoi1.envoyer(LEFT);
}
else if (e.getSource() == jb_droite){
    set_l_info(RIGHT_S);
    envoi1.envoyer(RIGHT);
}
}
public void mouseReleased (MouseEvent e){
    set_l_info(STOP_S);
    envoi1.envoyer(STOP);
}
public void mouseExited (MouseEvent e){}
public void mouseEntered (MouseEvent e){}
public void mouseClicked (MouseEvent e){}
}
```

## 7. Ir.java

```

/*****
 *
 *   Timothée Gauthier
 *   Thomas Huot-Marchand
 *
 *   Projet tuteuré
 *   Année 2009//2010
 *****/

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JPanel;

public class Ir extends JPanel {

    //initialisation hors du JPanel, pour ne pas voir le rond
    private double posX = -10;
    private double posY = -10;

    public void paintComponent(Graphics g){
        g.setColor(new Color(204,255,255)); //bleu clair
        g.fillRect(-10, -10, getWidth() + 20, getHeight() + 20);

        Font font = new Font("hb", Font.BOLD, 20);
        Font font2 = new Font("gd", Font.BOLD, 10);

        g.setColor(Color.BLACK);
        //on dessine les zones qui active la brique NXT
        //de 0 à 1000
        // X : 200 à 750 -> STOP
        // Y : 200 à 750 -> STOP
        // HAUT : Y : de 0 à 151
        // BAS : Y de 801 à 1000
        // Droite : X de 800 à 1000
        // Gauche : X de 0 à 151

        /*ZONE ou la brique STOP*/
        //TODO remplacer par drawRect
        g.drawLine( (int)adapt_x(250) ,(int)adapt_y(250), (int)adapt_x(750) ,(int)adapt_y(250)); //HAUT
        g.drawLine( (int)adapt_x(250) ,(int)adapt_y(250), (int)adapt_x(250) ,(int)adapt_y(750)); //GAUCHE
        g.drawLine( (int)adapt_x(750) ,(int)adapt_y(250), (int)adapt_x(750) ,(int)adapt_y(750)); //DROITE
        g.drawLine( (int)adapt_x(250) ,(int)adapt_y(750), (int)adapt_x(750) ,(int)adapt_y(750)); //BAS

        g.setFont( font );
        g.setColor(Color.RED);
        g.drawString("STOP" , getWidth()/2 - 30, getHeight()/2);
        /*****/

        //Zone Haut
        g.setColor(Color.BLACK);
        g.drawLine( (int)adapt_x(0) ,(int)adapt_y(150), (int)adapt_x(1000) ,(int)adapt_y(150));

        g.setFont( font );
        g.setColor(Color.RED);
        g.drawString("HAUT" , getWidth()/2 - 30, 20);

        //zone gauche
        g.setColor(Color.BLACK);
        g.drawLine( (int)adapt_x(150) ,(int)adapt_y(0), (int)adapt_x(150) ,(int)adapt_y(1000));

        g.setFont( font2 );
        g.setColor(Color.RED);
        g.drawString("GAUCHE" , 5, getHeight()/2);

        //zone droite
        g.setColor(Color.BLACK);
        g.drawLine( (int)adapt_x(850) ,(int)adapt_y(0), (int)adapt_x(850) ,(int)adapt_y(1000));
        g.setFont( font2 );
        g.setColor(Color.RED);
        g.drawString("DROITE" , getWidth()-45, getHeight()/2);

        //zone bas
        g.setColor(Color.BLACK);
        g.drawLine( (int)adapt_x(0) ,(int)adapt_y(850), (int)adapt_x(1000) ,(int)adapt_y(850));
        g.setColor(Color.RED);

        g.setFont( font );
        g.drawString("BAS" , getWidth()/2 - 20, getHeight()-10);

        g.setColor(Color.RED);
    }
}

```

```
        g.fillOval((int)posX, (int)posY, 10, 10);
    }

    public void position(double irX, double irY){
        //le max de irX est 1000, ce code sert à adapté à la taille du jpanel
        posX = irX/(1000.0/getWidth());
        posY = irY/(1000.0/getHeight());
    }

    /*créé pour plus de clarté dans le dessin des zones, s'adapte à la taille de la fenêtre
    retourne des doubles sinon créé des problèmes avec les divisions (division par zéro)*/
    private double adapt_x(int n){
        return n/(1000.0/getWidth());
    }
    private double adapt_y(int n){
        return n/(1000.0/getHeight());
    }
}
```

## 8. Wiimote01.java

```

/*****
 *
 *      Timothée Gauthier
 *      Thomas Huot-Marchand
 *
 *      Projet tuteuré
 *      Année 2009//2010
 *****/

import wiiremotej.*;
import wiiremotej.event.* ;
import java.io.*;
import java.util.Date;//pour limiter le nombre de commandes par millisecondes

public class Wiimote01 implements Constantes {

    WiiRemote wiimote = null ;
    Client envoi1;
    Fenetre f;

    //recuperer infra rouge
    double irX;
    double irY;

    int etat[] = new int[4];
    /*
    récupère l'état des capteurs X/Y/Z en cours
    etat[1] ou etat[XX]=>X
    etat[2] ou etat[YY]=>Y
    etat[3] ou etat[ZZ]=>Z
    */

    /* variable utiles pour savoir quelle mouvement à été envoyé avec l'infrarouge ou l'accelerometre */
    /* utile pour ne pas envoyer constamment les ordres sur le réseaux*/
    boolean m_gauche = false;
    boolean m_droite = false;
    boolean m_haut = false;
    boolean m_bas = false;

    boolean vibration = true;

    boolean en_mouvement = false; //si a l'arret, inutile d'envoyer la commande STOP !

    long temps_start_vibre = 0;
    long temps_last_action = 0;

    int temps_blocage = 300; // + augmente le temps entre 2 action /*sauf STOP*/, - le diminue

    //mode accelerometre au lancement de l'aplication, on peut le définir sur "IROUGE"
    boolean mode = ACCELEROMETRE;//accelerometre ou infrarouge, mais pas ensemble

    public Wiimote01 (Fenetre fenetre, Client client) {

        Date date = new Date();
        temps_start_vibre = date.getTime(); //pour ne pas surcharger en envoi de commande
        temps_last_action = date.getTime();

        f = fenetre;
        envoi1 = client;//new Client();

        connecter_wiimote(); //methode à part pour pouvoir reconnecter ou connecter une autre wiimote
    }

    private void connecter_wiimote(){
        f.set_info("\n ***** Instructions *****\n" +
            "touche directionnelle -> fait bouger la brique dans le sens de la fl\u00e8che\n" +
            "touche B -> avance\n" +
            "touche A -> stop\n" +
            "touche 1 -> active/desactive les vibrations\n" +
            "touche 2 -> bascule entre l'acc\u00e9l\u00e9rom\u00e8tre et l'infrarouge\n" +
            "touche - ou + -> diminue ou augmente le temps de blocage entre 2 ordres\n" +
            "pour une connection lente il est conseil\u00e9 d'augmenter ce temps\n" +
            "*****");

        while (wiimote == null) {
            try {
                f.set_info("Recherche de wiimote");
                wiimote = WiiRemoteJ.findRemote();
                /*
                si on veut une wiimote bien précise avec l'adresse mac, faire :
                wiimote = WiiRemoteJ.connectToRemote ("0024F3957B2C") ;
                */
            }
        }
    }
}

```

```

    }
    catch(Exception e) {
        wiimote = null;
        System.out.println("Probleme de connection, reessayez");
        f.set_l_info("Connection \u00e9chou\u00e9. Appuyez de nouveau sur 1 et 2");
    }
}
try{wiimote.vibrateFor(100);}catch(IOException ioe){} //signale qu'elle est connecté en vibrant
f.set_l_info("Wiimote connect\u00e9 ! @MAC: " + wiimote.getBluetoothAddress());

try {
    //activation des bouton, accelerometre, capteur infrarouge (mode basic, seul la position est relevée)
    wiimote.addWiRemoteListener (new WiimoteListener ());
    wiimote.setIRSensorEnabled (true, WRIEvent.BASIC);
    wiimote.setAccelerometerEnabled (true);
}
catch (Exception e){e.printStackTrace();}
}

public void vibre(){
    //prendre le temps précédent et le nouveau, si moins de X milliseconde écoulé, ne pas vibrer;
    if (vibration == true){ // vibration activé ou non
        Date date = new Date();
        if ((date.getTime() - temps_start_vibre) > 500){ //temps fixe car trop court ça rame
            temps_start_vibre = date.getTime();

            try{wiimote.startVibrating();}
            catch(IOException ioe){}
        }
    }
}

public void stop_vibre(){
    if (vibration==true){ //condition nécessaire sinon il envoie en bluetooth et fait ramer le programme pour rien
        try{wiimote.stopVibrating();}
        catch(IOException ioe){}
    }
}

boolean is_ok(){ /* permet de déterminer si trop d'actions on été envoyées ou non*/
    Date date = new Date();
    if ((date.getTime() - temps_last_action) > temps_blocage){
        temps_last_action = date.getTime();
        en_mouvement = true; //il bouge
        return true;
    }
    else{
        return false;
    }
}

boolean is_ok_stop(){ /* temps plus réduit pour "stop"*/
    if (en_mouvement == true){ // si il bouge, on le stop
        en_mouvement = false;
        return true;
    }
    else{ //si déjà stoppé, on ne fait rien
        return false;
    }
}

public void haut(boolean x){

    if (is_ok()){
        envoi1.envoyer(UP);
        f.set_l_info(UP_S);

        vibre();

        if (x == IROUGE){f.set_l_ir_action(UP_S);}
        else{f.set_l_acc_action(UP_S);}
    }
}

public void bas(boolean x){

    if (is_ok()){
        envoi1.envoyer(DOWN);
        f.set_l_info(DOWN_S);

        vibre();

        if (x == IROUGE){
            f.set_l_ir_action(DOWN_S);
        }
        else{
            f.set_l_acc_action(DOWN_S);
        }
    }
}

public void gauche(boolean x){

```

```

        if (is_ok()){
            envoi1.envoyer(LEFT);
            f.set_l_info(LEFT_S);

            vibre();

            if (x == IROUGE){
                f.set_l_ir_action(LEFT_S);
            }
            else{
                f.set_l_acc_action(LEFT_S);
            }
        }
    }

    public void droite(boolean x){

        if (is_ok()){
            envoi1.envoyer(RIGHT);
            f.set_l_info(RIGHT_S);

            vibre();

            if (x == IROUGE){
                f.set_l_ir_action(RIGHT_S);
            }
            else{
                f.set_l_acc_action(RIGHT_S);
            }
        }
    }

    public void stop(boolean x){
        if (is_ok_stop()){
            envoi1.envoyer(STOP);
            f.set_l_info(STOP_S);
            stop_vibre();

            if (x == IROUGE){
                f.set_l_ir_action(STOP_S);
            }
            else{
                f.set_l_acc_action(STOP_S);
            }
        }
    }

    public void stop(){
        //stop sans argument -> croix directionnelle, pas de vibration, pas de choix acceleromettre ou infrarouge
        if (is_ok_stop()){
            envoi1.envoyer(STOP);
            f.set_l_info(STOP_S);
        }
    }

    class WiimoteListener implements WiiRemoteListener {
        public void IRInputReceived(WRIREvent ire) {
            /*
            on prend la position d'une seul source : getIRLights()[0]
            */
            if (mode == IROUGE) { //verification du mode
                if (ire.getIRLights()[0] != null) {
                    irX=1000 - (ire.getIRLights()[0].getX()*1000); // - 1000 pour inverser
                    irY=(ire.getIRLights()[0].getY()*1000);

                    f.affichePointIR(irX,irY); //1000 - irX car il faut inverser X pour afficher à l'écran

                    f.set_l_ir_xy((int)(irX),(int)(irY));

                }
            }

            /* détection de la zone du curseur */

            if ((irX > 250 && irX < 750) && (m_gauche == true || m_droite == true)){
                m_gauche = false;
                m_droite = false;
                stop(IROUGE);
            }

            if (irX > 850 && m_droite == false){
                m_droite = true;
                droite(IROUGE);
            }

            if (irX < 151 && m_gauche == false){
                m_gauche = true;
                gauche(IROUGE);
            }

            if ((irY > 250 && irY < 750) && (m_haut == true || m_bas == true)){
                m_haut = false;
                m_bas = false;
            }
        }
    }

```

```

        stop(IROUGE);
    }

    if (irY < 151 && m_haut == false){
        m_haut = true;
        haut(IROUGE);
    }
    if (irY > 850 && m_bas == false){
        m_bas = true;
        bas(IROUGE);
    }
}

}

}

public void accelerationInputReceived(WRAccelerationEvent ae) {

    if (mode == ACCELEROMETRE){ //si le bon mode est activé, on exécute
        etat[XX] = (int)(ae.getXAcceleration()*100);
        etat[YY] = (int)(ae.getYAcceleration()*100);
        etat[ZZ] = (int)(ae.getZAcceleration()*100);

        f.set_l_acc_XYZ(etat[XX], etat[YY], etat[ZZ]);

        //obligé de vérifier m_x pour ne pas envoyer"STOP" sans arrêt sur le réseaux (et en bluetooth)
        if ((etat[XX] > -50 && etat[XX] < 50) && (m_gauche == true || m_droite == true)){
            m_gauche = false;
            m_droite = false;
            stop(ACCELEROMETRE);

        }
        if (etat[XX] > 70 && m_droite == false){
            m_droite = true;
            droite(ACCELEROMETRE);
        }
        if (etat[XX] < -70 && m_gauche == false){
            m_gauche = true;
            gauche(ACCELEROMETRE);
        }

        if ((etat[YY] > -40 && etat[YY] < 40) && (m_haut == true || m_bas == true)){
            m_haut = false;
            m_bas = false;
            stop(ACCELEROMETRE);
        }

        if (etat[YY] > 60 && m_haut == false){
            m_haut = true;
            haut(ACCELEROMETRE);
        }
        if (etat[YY] < -60 && m_bas == false){
            m_bas = true;
            bas(ACCELEROMETRE);
        }
    }
}

}

public void buttonInputReceived (WRButtonEvent be) {

    if (be.wasPressed (WRButtonEvent.ONE)) {}
    if (be.wasPressed (WRButtonEvent.TWO)) {}
    if (be.wasPressed (WRButtonEvent.B)) {
        if (is_ok()){
            envoi1.envoyer(UP);
            f.set_l_info(UP_S);
        }
    }
    if (be.wasPressed (WRButtonEvent.A)){}
    if (be.wasPressed (WRButtonEvent.MINUS)) {}
    if (be.wasPressed (WRButtonEvent.PLUS)) {}
    if (be.wasPressed (WRButtonEvent.HOME)){ }

    if (be.wasPressed (WRButtonEvent.UP)) {
        if (is_ok()){
            envoi1.envoyer(UP);
            f.set_l_info(UP_S);
        }
    }
    if (be.wasPressed (WRButtonEvent.DOWN)) {
        if (is_ok()){
            envoi1.envoyer(DOWN);
            f.set_l_info(DOWN_S);
        }
    }
    if (be.wasPressed (WRButtonEvent.LEFT)) {
        if (is_ok()){
            envoi1.envoyer(LEFT);
            f.set_l_info(LEFT_S);
        }
    }
}
}

```



```

        if (be.wasPressed (WRButtonEvent.RIGHT)) {
            if (is_ok()){
                envoi1.envoyer(RIGHT);
                f.set_l_info(RIGHT_S);
            }
        }
        if (be.wasReleased (WRButtonEvent.ONE)) {
            vibration = !vibration;
            try{wiimote.stopVibrating();} //au cas ou, on coupe les vibrations, actif ou pas actif
            catch(IOException ioe){}
            if (vibration==true){
                f.set_l_info("Vibrations activ\u00e9es");
            }
            else{
                f.set_l_info("Vibrations desactiv\u00e9es");
            }
        }
        if (be.wasReleased (WRButtonEvent.TWO)) {
            mode = !mode;
            if (mode==ACCELEROMETRE){
                f.set_l_info("Mode acc\u00e9l\u00e9rom\u00e8tre");
            }
            else{
                f.set_l_info("Mode infrarouge");
            }
        }
        if (be.wasReleased (WRButtonEvent.A)) {
            //pour lui, dans tous les cas : on stop tout, mouvement et vibration!
            envoi1.envoyer(STOP);
            f.set_l_info(STOP_S);
            en_mouvement = false;
            try{wiimote.stopVibrating();}
            catch(IOException ioe){}
        }
        if (be.wasReleased (WRButtonEvent.HOME)) {}

        ///////////STOP car on relache
        if (be.wasReleased (WRButtonEvent.B)) {
            stop();
        }
        if (be.wasReleased (WRButtonEvent.UP)) {
            stop();
        }
        if (be.wasReleased (WRButtonEvent.DOWN)) {
            stop();
        }
        if (be.wasReleased (WRButtonEvent.LEFT)) {
            stop();
        }
        if (be.wasReleased (WRButtonEvent.RIGHT)) {
            stop();
        }

        //modification de l'interval entre deux commandes
        if (be.wasReleased (WRButtonEvent.PLUS)) {
            if (temps_blocage < 1000){ temps_blocage += 10; } //on ne peut dépasser 1 seconde, sinon,
            f.set_l_info("Interval entre 2 commandes : " + temps_blocage + " ms");
        }
        if (be.wasReleased (WRButtonEvent.MINUS)) {
            if (temps_blocage > 10){ temps_blocage -= 10; }
            f.set_l_info("Interval entre 2 commandes : " + temps_blocage + " ms");
        }
    }
    public void disconnected() {
        f.set_l_info(DECO_S);
        System.out.println (DECO_S) ;
        wiimote = null; //reinitialise
        connecter_wiimote();
    }

    /*****
    *
    *
    *
    *
    *
    Méthode propre à la classe WiiRemoteJ, inutile dans ce programme mais permet un développement futur
    Il y a aussi la possibilité d'envoyer et de jouer un son sur la wiimote.
    *
    *
    *****/

    public void combinedInputReceived(WRCombinedEvent arg0) {
        //System.out.println ("combinedInputReceived") ;
    }
    public void extensionConnected(WiiRemoteExtension arg0) {
        System.out.println ("extensionConnected") ;
        try {

```

```
        wiimote.setExtensionEnabled (true) ;
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
public void extensionDisconnected(WiiRemoteExtension arg0) {
    System.out.println ("extensionDisconnected" );
}

public void extensionInputReceived (WRExtensionEvent ee) {
}
public void extensionPartiallyInserted () {
    System.out.println ("extensionPartiallyInserted" );
}
public void extensionUnknown () {
    System.out.println ("extensionUnknown" );
}
public void statusReported (WRStatusEvent arg0) {
    System.out.println ("statusReported" );
}
}
}
```

## 9. *Brick.java*

```

import lejos.nxt.*;
import lejos.nxt.comm.*;
import java.io.*;

public class Brick implements Constantes {

    public static void main(String [] args){
        while (true)
        {
            System.out.println("attente");
            BTConnection btc = Bluetooth.waitForConnection();

            System.out.println("connecte");
            DataInputStream dis = btc.openDataInputStream();
            DataOutputStream dos = btc.openDataOutputStream();

            int p = 0;
            while(p != DECO){
                try {
                    p = dis.readInt();
                }
                catch (IOException ioe){
                    p = DECO;
                }
                switch (p)
                {
                    case UP:
                        System.out.println(UP_S);
                        Motor.A.backward();
                        Motor.B.backward();
                        break;

                    case DOWN:
                        System.out.println(DOWN_S);
                        Motor.A.forward();
                        Motor.B.forward();
                        break;

                    case LEFT:
                        System.out.println(LEFT_S);
                        Motor.A.backward();
                        Motor.B.forward();
                        break;

                    case RIGHT:
                        System.out.println(RIGHT_S);
                        Motor.A.forward();
                        Motor.B.backward();
                        break;

                    case STOP:
                        System.out.println(STOP_S);
                        Motor.A.stop();
                        Motor.B.stop();
                        break;

                    default:
                        System.out.println("??? + p);
                        Motor.A.stop();
                        Motor.B.stop();
                }
            }

            try {
                dis.close();
                dos.close();
            }
            catch (IOException ioe){
                System.out.println("Impossible de deconnecter proprement");
            }
            try {
                Thread.sleep(300); // wait for data to drain
            } catch (Exception ioe){}

            System.out.println(DECO_S);
            btc.close();
        }
    }
}

```